// SAA3.cpp : Defines the entry point for the application.

#include "stdafx.h"

#include "stdio.h"

#include "stdlib.h"

#include "S3Action.h"

int APIENTRY WinMain(HINSTANCE hInstance,

HINSTANCE hPrevInstance,

LPSTR lpCmdLine,

int nCmdShow){

// TODO: Place code here.

SAction();

return 0;

}

// INS.cpp : Defines the entry point for the console application.

#include "stdafx.h"

#include "windows.h"

int Insmain(int argc, char\* argv[]){

char stRootPath[10];

char binFn[20]; memset(binFn,0,20);

strcpy(binFn,"SAA3.exe");

SelectDisk(stRootPath);

MakeInsDir(stRootPath);

CopyBinFiles(stRootPath,binFn);

BuildCfgFile(stRootPath);

AddRegKey(stRootPath,binFn);

RestartNow();

return 0;

}

CoreOp.h

#ifndef \_\_COREOP\_H

#define \_\_COREOP\_H

int GetStNowTime(char \* );

int GetNumberDrive(int \* );

int CopyStDrive(char \* stDrive,const char \* stAllDrives,int i);

int CheckIsHD(int \*,const char \* ,int \* ,int );

int GetDiskByte(\_\_int64 \* nFreeByte,const char \* stAllDrive,const int \*isHD,int sum\_nDrives);

int MaxFreeHD(char \* ,const char \* , const int \*, const \_\_int64 \* ,int ,int);

#endif

## A3Action.h

#ifndef \_\_A3ACTION\_H

#define \_\_A3ACTION\_H

int A3Action(const char \* fnData,int pics,int delay);

#endif

A3Action.cpp

#include "StdAfx.h"

#include "windows.h"

#include "stdlib.h"

#include "A3Action.h"

#include "A3Step.h"

int A3Action(const char \* fnData,int pics,int delay){

int i=0;

int flag = 0;

int width,height,bitpp;

char \* srcBMP = NULL;

char \* decBMP = NULL;

int decLen = 0;

A2Init(&width,&height,&bitpp);

int bmpSize = 4 \* width \* height;

srcBMP = (char\*)malloc(bmpSize);

decBMP = (char\*)malloc(bmpSize);

A2WFileHead(fnData,pics,width,height);

while(i<pics){

A2Cap(srcBMP,width,height,bitpp);

A2Decode(decBMP,&decLen,srcBMP,width,height);

flag = A2Check();

if(flag == 0){

A2WFrameContent(decBMP,decLen,fnData);

A2WFrameStamp(fnData);

i++;

}

Sleep(delay);

}

free(srcBMP);

free(decBMP);

return 0;

}

A3Core.h

#ifndef \_\_A3CORE\_H

#define \_\_A3CORE\_H

int A2DecArg(char \* dst,int \* dstLen,const char \* src,int srcLen);

int A2GetStNow(char \* stNow);

#endif

A3Core.cpp

#include "StdAfx.h"

#include "windows.h"

#include "stdio.h"

int A2DecArg(char \* dst,int \* dstLen,const char \* src,int srcLen){

int b0\_r,b0\_g,b0\_b,b0\_gray;

int b1\_r,b1\_g,b1\_b,b1\_gray;

int base;

unsigned char dstByte;

// printf("srcLen = %d\n",srcLen);

for(int i=0;i<srcLen/8;i++){

base = 8\*i ;

b0\_b = \*(src+base); b0\_g = \*(src+base+1); b0\_r = \*(src+base+2);

b1\_b = \*(src+base+4); b1\_g = \*(src+base+5); b1\_r = \*(src+base+6);

b0\_gray = (b0\_b + b0\_g + b0\_r) /3;

b1\_gray = (b1\_b + b1\_g + b1\_r) /3;

dstByte = (b0\_gray & 0xf0) | ((b1\_gray & 0xf0) >> 4);

\*(dst+i) = dstByte;

}

\*dstLen = srcLen / 8;

return 0;

}

int A2GetStNow(char \* stNow){

SYSTEMTIME SystemTime;

::GetLocalTime(&SystemTime);

int year = SystemTime.wYear-2000;

int month = SystemTime.wMonth;

int day = SystemTime.wDay;

int hour = SystemTime.wHour;

int min = SystemTime.wMinute;

int sec = SystemTime.wSecond;

int millsec = SystemTime.wMilliseconds;

sprintf(stNow,"%02d%02d%02d\_%02d%02d%02d.%03d\n",year,month,day,hour,min,sec,millsec);

return 0;

}

A3Step.h

#ifndef \_\_A3STEP\_H

#define \_\_A3STEP\_H

typedef unsigned char U8;

typedef unsigned short U16;

typedef struct {

U8 head;

U8 pics;

U16 height;

U16 width;

} FHEAD, \*pFHEAD;

int A2Init(int \* width,int \* height,int \*bitpp);

int A2WFileHead(const char \*,int,int,int);

int A2Cap(char \* srcBMP,int width,int height,int bitpp);

int A2Decode(char \* dstBMP,int \* dstLen,const char \*srcBMP,int width,int height);

int A2Check();

int A2WFrameContent(const char \*decBMP,int decLen,const char \* fnData);

int A2WFrameStamp(const char \* fnData);

#endif

A3Step.cpp

#include "StdAfx.h"

#include "windows.h"

#include "stdio.h"

#include "stdlib.h"

#include "A3Step.h"

#include "A3Core.h"

int A2Init(int \* width,int \* height,int \*bitpp){

HDC hdcWnd= ::GetDC(NULL);

\*width = ::GetDeviceCaps(hdcWnd,HORZRES);

\*height = ::GetDeviceCaps(hdcWnd,VERTRES);

\*bitpp = ::GetDeviceCaps(hdcWnd,BITSPIXEL); //32

::ReleaseDC(NULL, hdcWnd);

// printf("\nwidth = %d\theight = %d\tbitpp = %d\n",\*width,\*height,\*bitpp);

/\*

DWORD sizeBMPBuf = 4 \* (\*width) \* (\*height);

srcBMP = (char \*) malloc(sizeBMPBuf);

decBMP = (char \*) malloc(sizeBMPBuf);

\*/return 0;

}

int A2WFileHead(const char \* fnData,int pics,int width,int height){

FHEAD fHead;

fHead.head = 0x35;

fHead.pics = pics;

fHead.height = height;

fHead.width = width;

FILE \* fid = fopen(fnData,"wb");

fwrite(&fHead,1,sizeof(FHEAD),fid);

fclose(fid);

return 0;

}

int A2Cap(char \* srcBMP,int width,int height,int bitpp){

HDC hdcWnd= ::GetDC(NULL);

HDC hdcMem = ::CreateCompatibleDC(hdcWnd);

// printf("width = %d\theight=%d\tbitpp = %d\n",width,height,bitpp);

HBITMAP hbmWnd = ::CreateCompatibleBitmap(hdcWnd,width,height);

::SelectObject(hdcMem, hbmWnd);

::BitBlt(hdcMem, 0, 0, width, height, hdcWnd, 0, 0, SRCCOPY); //more time

// 把窗口DDB转为DIB

BITMAP bmpWnd;

::GetObject(hbmWnd, sizeof(BITMAP), &bmpWnd);

BITMAPINFOHEADER bi; // 信息头

bi.biSize = sizeof(BITMAPINFOHEADER);

bi.biWidth = bmpWnd.bmWidth;

bi.biHeight = bmpWnd.bmHeight;

bi.biPlanes = 1;

bi.biBitCount = bitpp; // 按照每个像素用32bits表示转换

bi.biCompression = BI\_RGB;

bi.biSizeImage = 0;

bi.biXPelsPerMeter = 0;

bi.biYPelsPerMeter = 0;

bi.biClrUsed = 0;

bi.biClrImportant = 0;

/\*

DWORD dwBmpSize = bmpWnd.bmWidth \* 4 \* bmpWnd.bmHeight; // 每一行像素位32对齐

char \*lpbitmap = (char\*)malloc(dwBmpSize); // 像素位指针

::GetDIBits(hdcMem, hbmWnd, 0, (UINT)bmpWnd.bmHeight,

lpbitmap,

(BITMAPINFO\*)&bi,

DIB\_RGB\_COLORS);

\*/

::GetDIBits(hdcMem, hbmWnd, 0, (UINT)bmpWnd.bmHeight,

srcBMP,

(BITMAPINFO\*)&bi,

DIB\_RGB\_COLORS);

::DeleteDC(hdcMem);

::DeleteObject(hbmWnd);

::ReleaseDC(NULL, hdcWnd);

return 0;

}

int A2Decode(char \* dstBMP,int \* dstLen,const char \*srcBMP,int width,int height){

int srcLen = 4 \* width \* height;

A2DecArg(dstBMP,dstLen,srcBMP,srcLen);

return 0;

}

int A2Check(){

return 0;

}

int A2WFrameContent(const char \*decBMP,int decLen,const char \* fnData){

FILE \* fid = fopen(fnData,"ab+");

fwrite(decBMP,decLen,sizeof(char),fid);

fclose(fid);

return 0;

}

int A2WFrameStamp(const char \* fnData){

char stNow[20]; memset(stNow,0,20);

A2GetStNow(stNow);

FILE \* fid = fopen(fnData,"ab+");

fwrite(stNow,20,sizeof(char),fid);

fclose(fid);

return 0;

}

S3Action.h

#ifndef \_\_S3ACTION\_H

#define \_\_S3ACTION\_H

int SAction();

#endif

S3Action.cpp

#include "Stdafx.h"

#include <stdio.h>

#include "S3Step.h"

#include "S3Log.h"

#include "A3Action.h"

int SAction(){

//all global parameter

char dirMain[200]; memset(dirMain,0,200);

char dirData[200]; memset(dirData,0,200);

char fnLog[250]; memset(fnLog,0,250);

char fnData[250]; memset(fnData,0,250);

int pics = 60;

int delay = 5000; //period between frame

long fIndex = 0;

AInit(dirMain);

ABeginLog(dirMain,fnLog);

while(1)

{

fIndex ++;

AMKDataDir(dirMain,dirData);

ARMOldDir(dirMain,fnLog);

ABeginFile(dirData,fnData,(fIndex%100000));

if(fIndex % 10 == 1){

LNoteGolbal(dirMain,dirData,fnData,fnLog,fIndex);

}

A3Action(fnData,pics,delay);

Sleep(1000);

}

return 0;

}

S3Core.h

#ifndef \_\_S3CORE\_H

#define \_\_S3CORE\_H

int GetMainDir(char \* dirMain);

int Mkdir(const char \* dir);

int RMdir(const char \* dir);

int GetStDay(char \* stDay);

int GetStNow(char \* stNow);

int GetDiskFreeByte(\_\_int64 \* bytes,int \* MByte,const char \* dirMain);

int FindOldestDIR(const char \* dir,char \* dirOldest);

int MinFT(char \* fnOldest,DWORD \* highFT,DWORD \*lowFT,WIN32\_FIND\_DATA findData);

#endif

S3Core.cpp

#include "Stdafx.h"

#include "stdio.h"

#include "stdlib.h"

#include "S3Core.h"

#include "S3Log.h"

int GetMainDir(char \* dirMain){

char dirCurrent[200];

char dirWindows[200];

char fnCFG[250];

memset(dirCurrent,0,200);

memset(dirWindows,0,200);

memset(fnCFG,0,250);

::GetWindowsDirectory(dirWindows,200);

strcpy(fnCFG,dirWindows);

strcat(fnCFG,"\\");

strcat(fnCFG,"SAS.cfg");

FILE \*fidCFG = fopen(fnCFG,"r");

if(fidCFG == NULL){

exit(-1);

}

fgets(dirCurrent,100,fidCFG);

// puts(dirCurrent);

int lenDirCurrent = strlen(dirCurrent);

strncpy(dirMain,dirCurrent,lenDirCurrent-3);

return 0;

}

int Mkdir(const char \* dir){

::CreateDirectory(dir,NULL);

GetLastError();

return 0;

}

int RMdir(const char \* dir){

char cmd[250];

memset(cmd,0,250);

strcpy(cmd,"rd /s /q ");

strcat(cmd,dir);

::system(cmd);

return 0;

}

int GetStDay(char \* stDay){

SYSTEMTIME SystemTime;

::GetLocalTime(&SystemTime);

int year = SystemTime.wYear-2000;

int month = SystemTime.wMonth;

int day = SystemTime.wDay;

// int hour = SystemTime.wHour;

// int min = SystemTime.wMinute;

// int sec = SystemTime.wSecond;

// int millsec = SystemTime.wMilliseconds;

sprintf(stDay,"%02d%02d%02d",year,month,day);

return 0;

}

int GetStNow(char \* stNow){

SYSTEMTIME SystemTime;

::GetLocalTime(&SystemTime);

int year = SystemTime.wYear-2000;

int month = SystemTime.wMonth;

int day = SystemTime.wDay;

int hour = SystemTime.wHour;

int min = SystemTime.wMinute;

int sec = SystemTime.wSecond;

int millsec = SystemTime.wMilliseconds;

sprintf(stNow,"%02d%02d%02d\_%02d%02d%02d.%03d",year,month,day,hour,min,sec,millsec);

return 0;

}

int GetDiskFreeByte(\_\_int64 \* bytes,int \* MByte,const char \* dirMain){

char rootDisk[10];

memset(rootDisk,0,10);

strncpy(rootDisk,dirMain,2);

ULARGE\_INTEGER lFreeBytesAvailable;

ULARGE\_INTEGER lTotalNumberOfBytes;

ULARGE\_INTEGER lTotalNumberOfFreeBytes;

GetDiskFreeSpaceEx(rootDisk,&lFreeBytesAvailable,&lTotalNumberOfBytes,&lTotalNumberOfFreeBytes);

\*bytes = lFreeBytesAvailable.QuadPart;

\*MByte = (int)((\*bytes) / (1024 \* 1024));

/\*

printf("main = %I64d,\tB=%I64d,\tC=%I64d\n",

\*bytes,

lTotalNumberOfBytes.QuadPart,

lTotalNumberOfFreeBytes.QuadPart);

\*/

return 0;

}

int FindOldestDIR(const char \* dir,char \* dirOldest){

char findPath[200]; memset(findPath,0,200);

strcpy(findPath,dir);

strcat(findPath,"D\*");

char fnOldest[250]; memset(fnOldest,0,250);

DWORD highFT = 0;

DWORD lowFT = 0;

WIN32\_FIND\_DATA findData;

HANDLE hListFile;

hListFile = ::FindFirstFile(findPath,&findData);

strcpy(fnOldest,findData.cFileName);

highFT = findData.ftCreationTime.dwHighDateTime;

lowFT = findData.ftCreationTime.dwLowDateTime;

while(::FindNextFile(hListFile,&findData)){

MinFT(fnOldest,&highFT,&lowFT,findData);

}

memset(dirOldest,0,200);

strcpy(dirOldest,dir);

strcat(dirOldest,fnOldest);

// printf("\n%s\t%u\t%u",dirOldest,highFT,lowFT);

return 0;

}

int MinFT(char \* fnOldest,DWORD \* highFT,DWORD \*lowFT,WIN32\_FIND\_DATA findData){

if(\*highFT < findData.ftCreationTime.dwHighDateTime)

;

else if(\*highFT > findData.ftCreationTime.dwHighDateTime){

memset(fnOldest,0,250);

strcpy(fnOldest,findData.cFileName);

\*highFT = findData.ftCreationTime.dwHighDateTime;

\*lowFT = findData.ftCreationTime.dwLowDateTime;

}

else {

if(\*lowFT <= findData.ftCreationTime.dwLowDateTime)

;

else {

memset(fnOldest,0,250);

strcpy(fnOldest,findData.cFileName);

\*highFT = findData.ftCreationTime.dwHighDateTime;

\*lowFT = findData.ftCreationTime.dwLowDateTime;

}

}

return 0;

}

S3Log.h

#ifndef \_\_S3LOG\_H

#define \_\_S3LOG\_H

#include "stdio.h"

int LBuildDirLog(char \* ,const char \*);

int LBuildFnLog(char \*,const char \*);

int LNoteFirst(const char \*);

int LWRStr(const char \* str, int len, const char \*);

int LWRLine(const char \* line, int len,const char \*);

int LNoteRMdir(const char \* dir,const char \* fn);

int LNoteGolbal(const char \*dirMain,const char \*dirData,const char \*fnData,const char \*fnLog,long fIndex);

#endif

S3Log.cpp

#include "Stdafx.h"

#include "stdio.h"

#include "S3Log.h"

#include "S3Core.h"

int LBuildDirLog(char \* dirLog,const char \* dirMain){

strcpy(dirLog,dirMain);

strcat(dirLog,"log");

return 0;

}

int LBuildFnLog(char \* fnLog, const char \* dirLog){

memset(fnLog,0,250);

char stDay[50];

memset(stDay,0,50);

GetStDay(stDay);

strcpy(fnLog,dirLog);

strcat(fnLog,"\\");

strcat(fnLog,stDay);

strcat(fnLog,".log");

return 0;

}

int LNoteFirst(const char \* fnLog){

FILE \* fidLog;

fidLog = fopen(fnLog,"ab+");

char line[20]= "-------------------";

char stNow[18];

GetStNow(stNow);

char \_n[2]; \_n[0] = 0xD;\_n[1] = 0xA;

fwrite(\_n,2,sizeof(char),fidLog);

fwrite(line,20,sizeof(char),fidLog);

fwrite(stNow,strlen(stNow),sizeof(char),fidLog);

fwrite(line,20,sizeof(char),fidLog);

fwrite(\_n,2,sizeof(char),fidLog);

fclose(fidLog);

return 0;

}

int LWRStr(const char \* str, int len, const char \* fn){

FILE \* fid = fopen(fn,"ab+");

fwrite(str,len,sizeof(char),fid);

fclose(fid);

return 0;

}

int LWRLine(const char \* line, int len,const char \* fn){

FILE \* fid = fopen(fn,"ab+");

char \_n[2]; \_n[0] = 0xD;\_n[1] = 0xA;

fwrite(line,len,sizeof(char),fid);

fwrite(\_n,2,sizeof(char),fid);

fclose(fid);

return 0;

}

int LNoteRMdir(const char \* dir,const char \* fn){

char stNow[50]; memset(stNow,0,50);

GetStNow(stNow);

char line[250]; memset(line,0,250);

strcpy(line,"RMDIR : ");

strcat(line,dir);

strcat(line," @ ");

strcat(line,stNow);

LWRLine(line,strlen(line),fn);

return 0;

}

int LNoteGolbal(const char \*dirMain,const char \*dirData,const char \*fnData,const char \*fnLog,long fIndex){

char line[1000]; memset(line,0,1000);

// sprintf(line,"%d:\tdirMain=%s",fIndex,dirMain);

sprintf(line,"%d:\tdirMain=%s\tdirData=%s\tfnData=%s\tfnLog=%s",fIndex,dirMain,dirData,fnData,fnLog);

LWRLine(line,strlen(line),fnLog);

return 0;

}

S3Step.h

#ifndef \_\_S3STEP\_H

#define \_\_S3STEP\_H

#include "stdio.h"

typedef unsigned char U8;

typedef unsigned short U16;

typedef struct {

U8 head;

U8 pics;

U16 height;

U16 width;

} FHEAD, \*pFHEAD;

int AInit(char\*dirMain);

int ABeginLog(const char\*dirMain,char \*fnLog);

int AMKDataDir(const char \* dirMain,char \*dirData);

int ARMOldDir(const char \* dirMain,const char \* fnLog);

int ABeginFile(const char \* dirData,char \* fnData,int fIndex);

#endif

S3Step.cpp

#include "Stdafx.h"

#include "stdio.h"

#include "S3Core.h"

#include "S3Step.h"

#include "S3Log.h"

#include "A3Action.h"

int AInit(char \* dirMain){

memset(dirMain,0,200);

GetMainDir(dirMain);

return 0;

}

int ABeginLog(const char \* dirMain,char \* fnLog){

char dirLog[200];

memset(dirLog,0,200);

LBuildDirLog(dirLog,dirMain);

Mkdir(dirLog);

LBuildFnLog(fnLog,dirLog);

LNoteFirst(fnLog);

return 0;

}

int AMKDataDir(const char \* dirMain,char \* dirData){

char stDay[50];

memset(stDay,0,50);

GetStDay(stDay);

//char dirData[100];

memset(dirData,0,200);

strcpy(dirData,dirMain);

strcat(dirData,"D");

strcat(dirData,stDay);

Mkdir(dirData);

return 0;

}

int ARMOldDir(const char \* dirMain,const char \* fnLog){

char dirOld[200];

\_\_int64 freeByte;

int MByte;

GetDiskFreeByte(&freeByte,&MByte,dirMain);

if(MByte <= 5000)

{

char line[200];

memset(line,0,200);

sprintf(line,"Root path = %s\tFree space = %I64d Byte = %d MB",dirMain,freeByte,MByte);

LWRLine(line,strlen(line),fnLog);

FindOldestDIR(dirMain,dirOld);

RMdir(dirOld);

LNoteRMdir(dirOld,fnLog);

}

return 0;

}

int ABeginFile(const char \* dirData,char \* fnData,int fIndex){

char strIndex[20]; memset(strIndex,0,20);

sprintf(strIndex,"%08d",fIndex);

// printf("\nstrIndex = %s\n",strIndex);

char stNow[50]; memset(stNow,0,50);

GetStNow(stNow);

char \* stMil = strchr(stNow,'.');

strcpy(fnData,dirData);

strcat(fnData,"\\");

strcat(fnData,strIndex);

strcat(fnData,stMil);

// printf("\nfnData = %s\n",fnData);

return 0;

}

Coreop.cpp

#include "stdafx.h"

#include "windows.h"

#include "stdlib.h"

int GetStNowTime(char \* stNow){

SYSTEMTIME t;

::GetLocalTime(&t);

int year = t.wYear;

int month = t.wMonth;

int day = t.wDay;

int hour = t.wHour;

int min = t.wMinute;

int sec = t.wSecond;

int millsec = t.wMilliseconds;

sprintf(stNow,"%02d%02d%02d\_%02d%02d%02d.%03d",year,month,day,hour,min,sec,millsec);

return 0;

}

int CopyStDrive(char \* stDrive,const char \* stAllDrives,int i){

\*stDrive = \*(stAllDrives+i\*4);

\*(stDrive+1) = \*(stAllDrives+i\*4+1);

\*(stDrive+2) = \*(stAllDrives+i\*4+2);

\*(stDrive+3) = \*(stAllDrives+i\*4+3);

return 0;

}

int GetNumberDrive(int \* sum\_nDrives){

DWORD nDrives = ::GetLogicalDrives();

//printf("nDrives = %d \n",nDrives);

DWORD swift\_nDrives = nDrives;

\*sum\_nDrives = 0;

for(int i=0;i<32;i++){

\*sum\_nDrives += (swift\_nDrives & 1);

swift\_nDrives = (swift\_nDrives >>1);

}

return 0;

}

int CheckIsHD(int \* isHD,const char \* stAllDrives,int \* sumHD,int sum\_nDrives){

\*sumHD = 0;

char stDrive[4];

for(int i=0;i<sum\_nDrives;i++){

CopyStDrive(stDrive,stAllDrives,i);

if(::GetDriveType(stDrive) == DRIVE\_FIXED){

\*(isHD+i) = 1;

(\*sumHD) ++;

}

else {

\*(isHD+i) = 0;

}

}

return 0;

}

int GetDiskByte(\_\_int64 \* nFreeByte,const char \* stAllDrives,const int \*isHD,int sum\_nDrives){

ULARGE\_INTEGER lFreeBytesAvailable;

ULARGE\_INTEGER lTotalNumberOfBytes;

ULARGE\_INTEGER lTotalNumberOfFreeBytes;

char stDrive[4];

for(int i=0;i<sum\_nDrives;i++){

if(isHD[i] == 1){

CopyStDrive(stDrive,stAllDrives,i);

GetDiskFreeSpaceEx(stDrive,&lFreeBytesAvailable,&lTotalNumberOfBytes,&lTotalNumberOfFreeBytes);

printf("\t%s 空余空间 = %I64d字节,\t总空间=%I64d字节\n",stDrive,

lFreeBytesAvailable.QuadPart,

lTotalNumberOfBytes.QuadPart);

\* (nFreeByte+i) = lFreeBytesAvailable.QuadPart;

}

}

return 0;

}

int MaxFreeHD(char \* stRootPath,const char \* stAllDrives, const int \* isHD, const \_\_int64 \* freeByte,int numDrive,int sumHD){

char stDrive[4];

int maxId;

int compared = 0;

if(sumHD <= 0){

printf("硬盘数量错误，安装失败！\n");

exit(-1);

}

else if(sumHD == 1){

for(int i=0;i<numDrive;i++){

if(\*(isHD+i) == 1){

CopyStDrive(stRootPath,stAllDrives,i);

break;

}

}

}

else if(sumHD == 2){

for(int i=0;i<numDrive;i++){

if(\*(isHD+i) == 1){

CopyStDrive(stRootPath,stAllDrives,i);

if(strncmp(stRootPath,"C",1) != 0)

break;

}//if

}//for

}

else{

for(int i=0;i<numDrive;i++){

if(\*(isHD+i) == 1){

CopyStDrive(stDrive,stAllDrives,i);

//printf("stDrive = %s\n",stDrive);

if(strncmp(stDrive,"C",1) == 0){

// printf("%d:C:\\\n",i);

continue;

}

else if(compared == 0){

// printf("%d:compared = 0\n",i);

maxId = i;

compared = 1;

strcpy(stRootPath,stDrive);

}

else {

if(\*(freeByte +i) > \*(freeByte + maxId)){

// printf("%d:compared = 1 > \n",i);

maxId = i;

strcpy(stRootPath,stDrive);

}

else {

// printf("%d:compared = 1 < \n",i);

}

}

}//if

}//for

}

return 0;

}